New Developments in EPOS Tools for Configuring and Generating Embedded Systems

Rafael L. Cancian, Marcelo R. Stemmer
Department of Automation and Systems (DAS)
Federal University of Santa Catarina (UFSC)
{cancian,marcelo}@das.ufsc.br

Antônio Augusto M. Fröhlich
Laboratory for Hardware and Software Integration (LISHA)
Computer Science Department (INE)
Federal University of Santa Catarina (UFSC)
guto@lisha.ufsc.br

Abstract

Embedded systems usually run dedicated applications in highly restricted environments. This paper describes our approach to configure and generate embedded systems and a tool to assist this process that is being used with EPOS (Embedded and Parallel Operating System), an OS developed using AOSD. This tool receives a hight level specification of the application and builds the necessary computational support for it. This paper describes some improvements to the tool that are enabling this process to be done automatically. Co-design, design space exploration and partitioning techniques were modeled as independent components, which allow to modify the implementation technique used in each step of this process and to use it to compose a new tool without changing the whole developing chain. Our main contribution is in the development of highly portable and application-oriented embedded systems using AOSD in a semi-automatized process guided by our tool.

1. Introduction

Embedded systems can be implemented over some different kind of hardware support, including microcontrollers and programmable logic devices (PLD). Microcontrollers have fixed hardware and allow software programming. They exist in several families and types and it's easy to find a type that internally aggregates the necessary hardware components. Even if this doesn't happen, external hardware components can be interconnected to the microcontroller using a printed circuit board to build the hardware platform. PLD are mainly represented by FPGAs. They have programmable hardware, what makes possible to include the necessary hardware components that matches exactly application requirements. The existence of tools for translating programming languages into hardware description languages (HLD), the possibility of synthesizing microcontrollers or processors as a hardware component (soft core) into a FPGA, and the advances in synthesis CAD tools, have allowed all computational support (sw and hw) to be placed into a single PLD, creating a System-on-a-Chip (SoC).

The advances in FPGA tecnology have produced a strong tendency in using that devices as hardware platform for embedded systems because they actually show some advantages, including: (i) facilities in integrating hardware components into a single chip, (ii) flexibility and CAD tools to support hardware development, (iii) possibility to co-design hardware and software, and (iv) partitioning system components into hardware or software domain. Microcontrollers have evolved too and continue to be very used. However, during the last decade and a half, research efforts were focused into the development of embedded systems as SoCs, MpSoCs ou Network of SoCs built over FPGAs.

This paper is organized as follow: Section 2 briefly discuss the methodologies for co-design and the AOSD, our approach to design embedded systems. Section 3 presents our tool used to assist the process of configuring and generating embedded systems. Section 4 finalizes with some conclusions and a roadmap for future work.

2 Fundamentals

2.1 Hardware/Software Co-design

Methods for hardware/software co-design were created as a solution for several problems related to the integration of software and hardware components that used to happen even when individual components work correctly following their logical specifications. These methods include tests and integration in every single step of the design, allow both (hw and sw) teams to work in parallel, avoid error propagation to the next step, and help to determine a combination of possible implementations for the components that compose the system, eg, hardware/software partitioning. In general, they differ in relation to (i) the abstraction level and input notation, (ii) the system model used, and (iii) the partitioning technique used.

Using a single description language to specify both hardware and software brings up several advantages to embedded system design. Much effort have been directed to the identification of basic blocks that could be mapped to any implementation domain (hw or sw). Despite all research and optimism, [5] from Xilix Research Labs, says that there's no actual compiler in position to translate algorithms in programming language into efficient hardware implementations for different architectures.While actual compilers do not allow efficient hardware synthesis from high level software programming languages and tools do not perform all design steps in an integrated and efficient way, the general architecture model approach will continue to be used for embedded systems design.

2.2 Hardware/Software Partitioning

Hardware/software partitioning is considered one of the most important steps in hardware/software co-design, because in this step several design alternatives are explored to find an optimal (or sub-optimal) solution that matches some design constraints (usually related to silicon area and latency). The partitioning is basically a combinatorial optimization problem. Interactive and automatic approaches have been proposed. The interactive approach requires the designer intervention and is very used because the performance estimation of the design alternatives depend on several aspects and is a complex problem. The automatic approach goals to minimize the cost-function used and its performance depends on the techniques used and on the cost-function representativity and accuracy. Techniques like Integer Linear Programming (ILP) [6], Simulated Annealing (SA) [1], Tabu Search [2], Genetic Computing [10] [7], Group Migration, and Hierarchical Clustering have been used.

The main problems with actual techniques seem to be related to cost models, and performance estimations, and to the representation of cost-functions to be minimized. Since system models are being expressed using even higher abstraction level, it has became too hard and even impossible to extract accurate performance estimations. Using less abstract models (eg RTL) is possible to get better estivatives. However, the process to produce and to simulate such models usually takes a prohibitive amount of time. As a consequence, the cost-functions usually doesn't represent an accurate realization of the system.

2.3 Design Space Exploration

The design space exploration aims at generating models for each project alternative and to obtain values to estimate their performance and to create the cost-function to be minimized by the partitioning algorithm. The choice of which project alternative will be evaluated to extract performance metrics is systematized to better explore the design space, and that's usually done by a design experiment approach. The system model, at the adequate abstraction level, is a compromise between accuracy and processing time. Higher abstraction level models are faster, but produce less accurated metrics or simply do not allow the extraction of some metrics (eg energy consumption).

Despite all advances and researches on the design of embedded system over PLD, using microcontrollers may be simply the best option in several cases. And this project alternative must be taken into account by tools that assist embedded system's designers. In this sense, the design space exploration should begin answering a more general question than what is the silicon area consumed or the latency of the future synthesized circuit: Would be better to implement that system over microcontrollers or PLD? This kind of design/architectural space exploration should precede traditional exploring and partitioning approaches, and should be based on more general features of the system, as the system final cost, for example.

2.4 AOSD and EPOS

The Application-Oriented Systems Design method (AOSD) proposes strategies to define components that represent significant entities in different domains. AOSD allows the modeling of independent abstractions and organizes them as family members, as defined in the Family-Based Design (FBD) [8]. To reduce environment dependences and to increase abstractions re-usability, AOSD aggregates to the decomposition process the main concern of Aspect-Oriented Programming (AOP): aspects separation. With the use of this concern, it is possible to identify scenario variations and non-functional properties and to model them as scenario aspects that crosscut the entire system. The integrated utilization of these and other advanced software engineering techniques allows the development of efficient methodologies for Embedded Systems Design, both in basic software and in hardware domains.

One of the first practical strategies using AOSD is the one proposed by [3], the EPOS (Embedded Parallel Operating System). EPOS is a framework conceived through AOSD that combines concerns of FBD, OAP, Object Oriented Design (OOD) and Static Meta Programming (SMP) to guide the development of scenario independent component families that, by applying scenario adapters, can be used in different environments and provide architecture transparency [4]. Besides operating system components, it has been extended to deal with hardware [9], allowing for the design of hybrid components whose software/hardware implementations are suitable. This approach has so far enabled the development of run-time support systems with architectures that are defined according to the particular needs of applications. Indeed, with all these features it seems a promising approach to help solving the problems that currently limit efficiency in embedded system development.

3. Developments in EPOS Tool

In this paper we present the general basic principles of the approach is being used in our research group to assist the process of configuring and generating embedded systems and also of the tool and techniques that are being included into the framework used in this process. A prototype implementation of this tool is in use and new advances are being included to help solving the problems mentioned in the last section. The development of such new advances is being guided by some desirable features identified by our research group while designing and developing some real embedded systems. According to our expertise, a tool to assist the design of embedded systems should include the following four features:

(1) To have a well designed and waste repository of portable and reusable software and hardware components. Components must be independent of execution scenario and need to be completely defined with dependences and composition rules that allow their automatic adaptation and composition, and features that allow their selection and partitioning. This feature is already implemented in our tool that can identify, select, adapt, and compose components. Current repository supports independent software and hardware components. New developments are defining and implementing an architecture for hybrid hardware/software components, that are software engineering artifacts that freely combine hardware and software elements.

(2) To support several co-design methods, including design space exploration, hardware/software partitioning, and components specified in different languages (C++, SystemC, VHDL, etc) and abstraction levels. Our tool was modeled to represent the design steps as components themselves. That means the implementation of a particular design step can be switched for another one without compromising the functionality of other design steps or tool modules. The support for C++ and VHDL is already implemented. Current work aims at providing support for SystemC components and for automated design space exploration and hw/sw partitioning techniques.

(3) To allow the designer to choose an interactive or automatic process on each design step. An interactive process gives the developer some choices and waits for his selection, thus better capturing his knowledge. An automatic process is based on previous designer choices or techniques that don't need human interference. Actually our tool accepts only the interactive process in all steps, except for the first one (the analysis of source-code). Current works are implementing automatic techniques for each step.

(4) To consider both microcontrollers and FPGAs as possible alternatives for target-platforms. Our approach uses Feature Based Modeling (FBM) instead of traditional costs (latency, silicon area, memory usage, etc). This allow architectural exploration (several microcontrollers and FPGAs based platforms can be evaluated as possible target platforms) and the use of more high level features. Other difference is that a feature is not characterized by a single value, but for a tuple allowing to indicate the knowledge or the incertitude about that feature.

Our approach relies on a static configuration mechanism that allows the generation of optimized versions of the operating system and hardware for each of the applications that are going to use it. This approach was implemented using EPOS framework and consists on a repository of hardware and software components, files to represent dependences, composition rules, scenario adapters, traits and features, and a tool that use all these stuff to configure and generate application-oriented embedded systems. The tool was divided into four major modules: `Analizer`, `Partitioner`, `Configurator`, and `Generator`.

The `Analyser` is responsible for identifying what features are required from the application, and elaborates a requirement specification that includes methods, types, and constants used by the application. This module seeks the input for references to the components' interfaces (methods that compose the OS API), what could be done based on high level input specifications of the system, such as UML or source-code. The actual implementation of the `Analyzer` assumes the input is the application source-code. It applies a technique that involves the compilation of the application's source code, a look at the resulting object files, and the identification of unresolved symbols (the EPOS API). It's useful to remember the tool modules were designed as independent components. It means that other implementation that reads a XMI file describing the application (with UML diagrams) could also be used to search for references to the components' interfaces and to elaborate a requirement specification, with no modifications to the tool chain. A component dependency tree is produced and used to feed the `Partitioner`. Multiple project alternatives are coded as alternative components (nodes) in such structure.

The description of components must be complete enough so that the `Partitioner` module will be able to automatically identify which abstractions better satisfy the requirements of the application without violating design requirements, generating conflicts or invalid configurations and compositions. A component is defined by a *family* and its set of *member*. In addiction to that, this enriched description can be used to perform design space exploration. A dependency tree with no alternative components corresponds to a unique project alternative and features are used to map how components meet design constraints. The combination of all possible projects, including possible target-platforms, forms the design space to be explored.

The description of the interfaces in a family and its members is the main source of information for the `Configurator`, but correctly assembling a component-based system goes far beyond the verification of syntactic interface conformance: non-functional and behavioral properties must also be conveyed. For this purpose, the component description language includes two special elements: `feature` and `dependency`. These elements can be applied to virtually any other element in the language to specify features provided by components and dependencies among components that cannot be directly deduced from their interfaces. Enriching the description of components with features and dependencies can significantly improve the correctness of the assembly process, helping to avoid inconsistent component arrangements.

In the last module, the `Generator` allows the designer to launch processes that invoke the operating system's makefiles, causing the system instance generation, and processes that invoke synthesis tools that build the hardware platform (if it's a FPGA). Also, the application may be compiled by the `Generator` with parameters that consider the system that was just built for it. Our approach aims at generating real systems, not only simulated ones. Possible implementations of this module could generate a system's model at different abstraction levels (co-simulation models) to provide performance metrics back to the `Partitoner` in an iterative process. A limitation of the actual implementation of the `Generator` is that it only generates the final system, composed by a software image and, depending on target-platform, also the bitstream file to configure the FPGA, but does not simulate the system or obtains performance metrics. Current developments are creating a new Generator component to provide such functionality.

Results

This tool has helping us to produce interesting results. Its strong points are its fast and simple operation and its internal design that features good maintainability and extensibility. It allows the developer to focus on the application and not on sw/hw support, thus significantly reducing development time. Several functional embedded systems were designed and fully prototyped using this tools. Examples include multimedia, digital tv, wireless sensor network and energy-aware applications. The final system is optimized for the target-application and is composed only by necessary and sufficient components. As an example, the software image for the Philosophers' dinners problem running on a IA32 is 42KB large, including boot, setup, OS and application.

4. Conclusions

In this paper we dealt with some problems of developing embedded systems. We have briefly presented a tool that assists developers in configuring and generating software and hardware support for embedded systems taking as base a collection of reusable components developed according with the Application-Oriented System Design methodology, their dependencies, composition rules and features. The prototype effectively identifies, selects, configures, adapts, and composes those components, generating real and functional embedded systems.

The techniques currently implemented do not represent a complete solution to automate the process of generating embedded systems whereas the designer still takes some decisions.However, that's not a limitation of the approach itself but only of the actual prototype implementation. Thus, work in progress includes the implementation of automated techniques for each module. Our main contribution is in the development of highly portable and application-oriented embedded systems using AOSD in a semi-automatized process guided by our tool. Working with higher level system features instead of components' costs we are getting us in position to completely automate this process and to explore the design space including the target-platform as one dimension of it.

References

[1] J. H. D. Herrmann and R. Ernst. An approach to the adaptation of estimated cost parameters in the cosyma system. In *International Conference on Hardware Software Codesign - Proceedings of the 3rd international workshop on Hardware/software co-design*, pages 100 – 107, 1994.

[2] P. E. et al. System level hardware/software partitioning based on simulated annealing and tabu search. In *Design Automation for Embedded Systems*, 1997.

[3] A. A. Fröhlich. *Application-Oriented Operating Systems*. PhD thesis, Sankt Augustin: GMD - Forschungszentrum Informationstechnik, 2001.

[4] A. A. Fröhlich and W. Schröeder-Preikschat. Scenario adapters: Efficiently adapting components. In *Proceedings of 4th World Multiconference on Systemics, Cybernetics and Informatics*, 2000.

[5] P. Lysaght. Xilinx technical report. Technical report, Xilinx Research Labs, 2003.

[6] R. Niemann and P. Marwedel. An algorithm for hardware/-software partitioning using mixed integer linear. In *Proceedings of the ED&TC*, 1996.

[7] G. Z. P. Mudry and G. Tempesti. Hybrid genetic algorithm for constrained hardware-software partitioning. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, pages 1 – 6, March 2006.

[8] D. L. Parnas. On the design and development of pro- gram families. In *IEEE Transactions on Software Engineering*, pages 1–9, 1976.

[9] F. V. Polpeta and A. A. Fröhlich. Hardware mediators: a portability artifact for component-based systems. In *Proceedings of International Conference on Embedded and Ubiquitous Computing*, volume 3207, pages 271–280, 2004.

[10] S. R. V. Srinivasan and R. Vemuri. Hardware software partitioning with integrated hardware design space exploration. In *Proceedings of Design, Automation and Test in Europe*, pages 28 – 35, February 1998.